intel®

ARTICLE REPRINT

AR-184

October 1981

Silicon Operating System Standardizes Software

C. McMinn, R. Markowitz, J. Wharton, and W. Grundmann

Electronics, September 8, 1981

Silicon operating system standardizes software

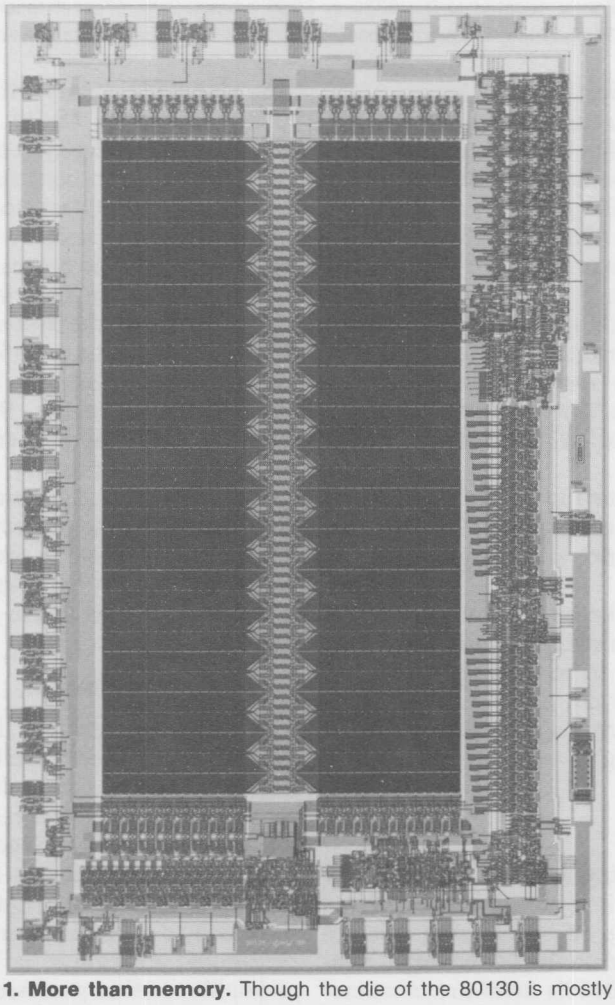
All the basic functions for real-time multitasking programs, including software subroutines and hardware timers, fit on the 80130

by C. McMinn, R. Markowitz, J. Wharton, and W. Grundmann, *Intel Corp., Santa Clara, Calif.*

☐ Silicon software, an inherently ambiguous phrase, refers to the solid-state realization of standard programs. It is the midpoint in the migration of microprocessor software into microcode and, as such, it exhibits the characteristics of both. Like software, it must be reconfigurable if it is to endure the fast-changing world of microprocessor technology, and like hardware, it must capitalize on this technology to reduce hardware and software costs and to provide increased system performance.

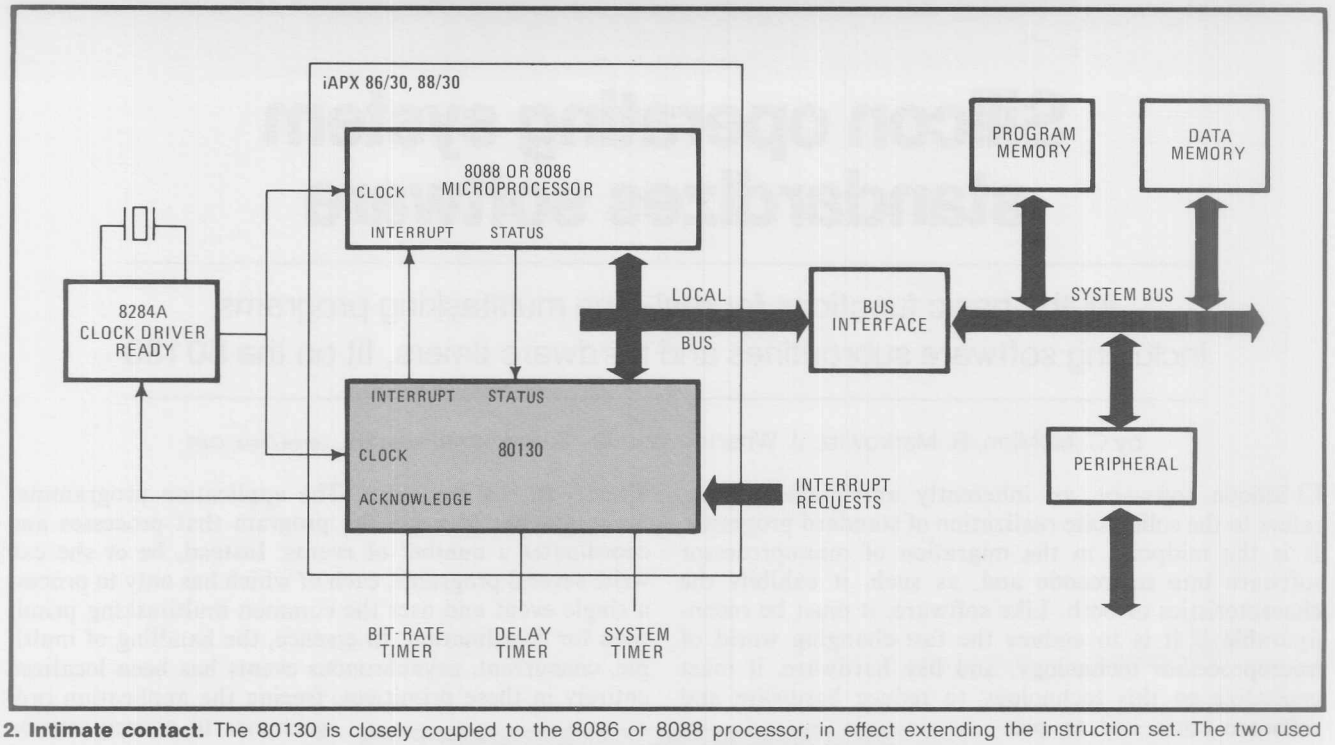
Recognizing this need, Intel Corp. has designed a set of silicon operating-system primitives that provides all of the basic building-blocks needed to write real-time multitasking software. These building blocks have been carefully chosen to include the functionality needed today as well as to allow reconfigurability for tomorrow. But the 80130 is not just software, since it also includes on chip all the other hardware components necessary to make the system work without additional external logic (Fig. 1). It is designed to be closely coupled to either the 8086 or 8088 16-bit microprocessor, creating the iAPX 86/30 and 88/30 two-chip sets.

These two-chip operating-system processors, as they are called, were specifically developed to remove the burden of designing multitasking operating-system primitives from the application programmer by providing a well-defined, fully debugged set implemented directly in the hardware. The application programmer no longer has to write the program that processes and coordinates a number of events. Instead, he or she can write several programs, each of which has only to process a single event and uses the common multitasking primitives for coordination. In essence, the handling of multiple, concurrent, asynchronous events has been localized entirely in these primitives, freeing the application programmer to concentrate on his or her own program and allowing an application to be developed faster and with reduced software costs.



1. More than memory. Though the die of the 80130 is mostly occupied by the 16-K-byte kernel control store, it also includes an interrupt controller and two 16-bit timers for operating-system use, plus a bit-rate generator for the user's convenience.

The primitives support an event-oriented design that entrusts the application to separate concurrent tasks, rather than burdening a single program with the complex dependencies inherent in random real-time events. Each event can be processed by a separate task or along with closely related events in a common task. External events and interrupts are processed by the interrupt-handling primitives directly from the on-chip interrupt controller subsystem as they occur in real time. Multiple tasks and multiple events are coordinated by the scheduler, whose preemptive, priority-based scheduling algorithm and system timers organize and monitor the processing of each task to guarantee that events are performed in their order of importance. The 80130 also provides primitives for intertask communication—by mailboxes—and for mutual exclusion—by regions—both of which are essential



2. Intimate contact. The 80130 is closely coupled to the 8086 or 8088 processor, in effect extending the instruction set. The two used together form the iAPX-86/30 or 88/30 operating-system processors. With an 8087 arithmetic processor, they form the 88/31.

functions for multitasking applications.

From the application programmer's viewpoint, the 80130 extends the base of the 8086 architecture by providing more than 30 operating-system primitive instructions (see Table 1), making the 80130 a logical and easy-to-use extension to an 8086 or 8088 system.

The 80130 replaces approximately 10 large- and small-scale integrated circuits in a system. It sits directly on the local bus of the processor and runs with both the 8086 and 8088 at an 8-megahertz clock rate. This guarantees that, regardless of the speed of the remaining memory in the system, these time-critical operating-system primitives will operate at the maximum bus bandwidth.

The chip has also been designed to be compatible with the iRMX-86 operating system and, as such, has been thoroughly tested against a wide range of text software available for this product.

The 80130 primitives were chosen after much analysis both because they are useful in many applications today and because they will continue to be useful primitives, if not standard machine instructions, in future processors. The 80130 kernel implementation is simple and efficient yet powerful enough to be a highly versatile architectural building block.

Architectural details

The 80130 is connected directly to the local bus of the 8086 and 8088 processor (it automatically detects whether the 8086 or 8088 is present), with address decoding, buffering, and bus-demultiplexing logic contained on chip (Fig. 2). The 80130's firmware is memory-mapped to any 16-K boundary in the processor's 1-megabyte address space. The control registers are mapped into the input/output space; they are aligned on any 16-byte boundary within the space.

Internally, the 80130 firmware consists of two sections: an operating-system unit and a control unit (Fig. 3). The former consists of a 16-K-byte operating-system–kernel control store complete with an operating-system timer and a delay timer; a bit-rate generator; and 8259A-compatible programmable interrupt logic.

The first timer generates the fundamental real-time clock period in the system. It is set to 10 milliseconds initially but can be modified by the system designer. The delay timer supports the kernel timing function by indicating the next event. Both these timer resources are reserved for use by the kernel.

The bit-rate generator, which has a range of 75 to 768 kilobits per second, is provided as a user resource. The 80130 interrupt logic vectors eight independent priority levels, one of which is reserved for the operating-system timers. Optional external slave interrupt controllers (8259A) can expand the number of user-programmable interrupt levels to 57.

Individual treatment

The 80130 interrupt logic goes beyond that of an 8259A interrupt controller. First, it allows the eight interrupt inputs to be individually programmed as level- or edge-sensitive (whereas the 8259A requires all to be programmed alike). Second, the 80130 has an output line that can be used in conjunction with the 8289 bus arbiter to reduce interrupt latency by localizing interrupt response to a single board in a multiboard system. An additional advantage is that the system bus remains available to other processors during an interrupt.

The 80130 supplements the 8086's basic architecture with five new objects, or system data types—jobs, tasks, segments, mailboxes, and regions.

TABLE 1 PRIMITIVES USED IN 80130 OPERATING SYSTEM FIRMWARE

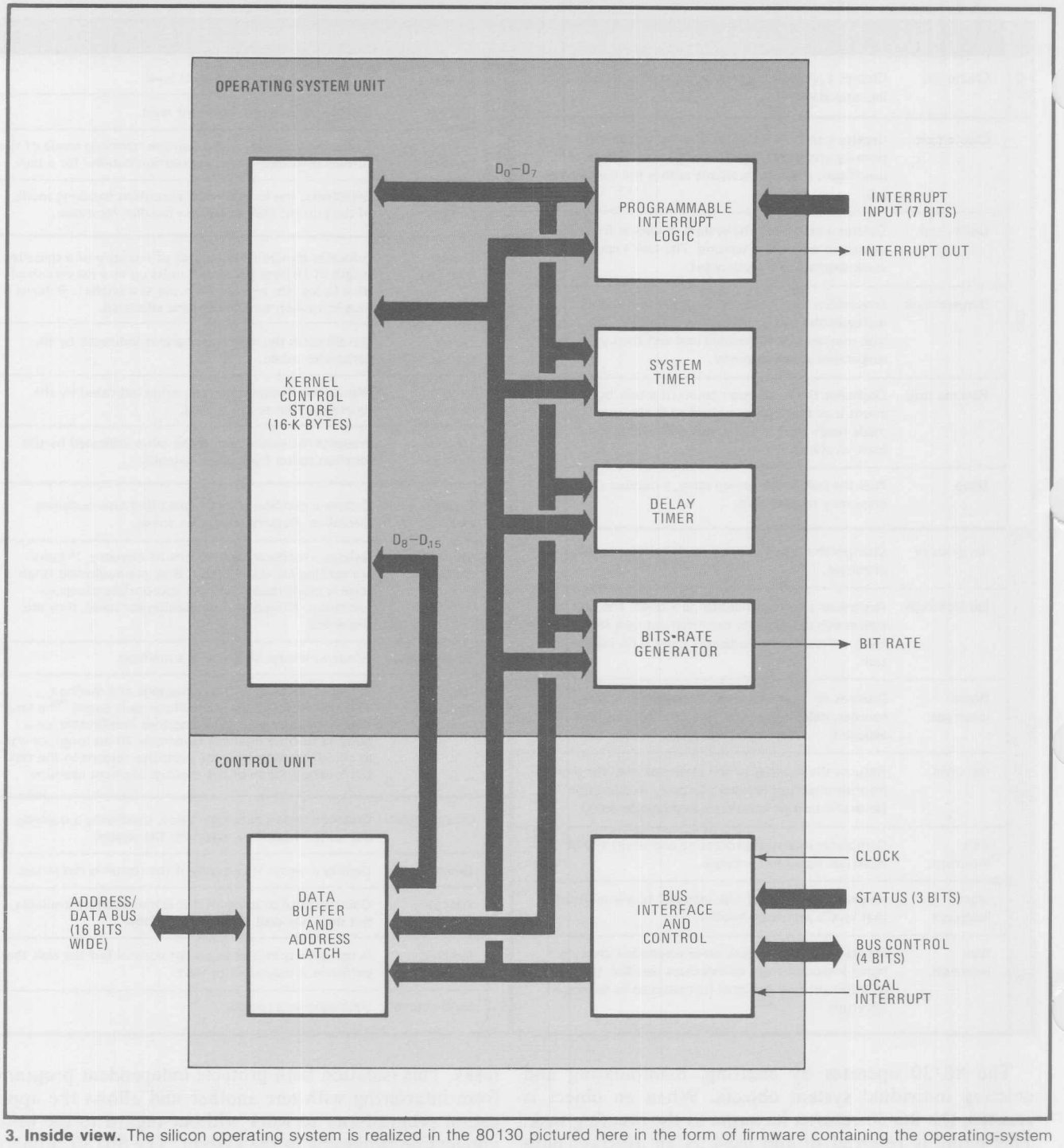
Job	Create job	Creates a job partition including memory pool, task list, and stack area.
Task	Create task	Creates a task with the specified environment and priority and puts it in the ready state. Checks for insufficent memory available within the containing job.
	Delete task	Deletes a task from the system as well as from any queues in which it is waiting. The task's state and stack segment are deallocated.
	Suspend task	Suspends a task (changes its status to suspended) or increases the task's suspension count by 1. A sleeping task may also be suspended and will then awaken suspended unless resumed.
	Resume task	Decreases the suspension count of a task by 1. If the count is at that point reduced to 0, the task state is made ready or if it was suspend-asleep, it is put back to asleep.
	Sleep	Puts the task in the asleep state, a number of 10-ms units may be specified.
Interrupt	Set priority	Changes the task's priority to the value passed in the primitive.
	Set interrupt	Assigns an interrupt handler to a level. The task that makes this call is made the interrupt task for the same level, unless the call indicates there is no interrupt task.
	Reset interrupt	Disables an interrupt level. Cancels the interrupt handler, deletes the interrupt task for that level if assigned.
	Get level	Returns the number of the interrupt level for highest-priority interrupt handler currently in operation (several interrupt handlers could be operating).
	Exit interrupt	Completes interrupt processing and sends end-of-interrupt signal to hardware.
	Signal interrupt	Invokes the interrupt task assigned to a level from that level's interrupt handler.
	Wait interrupt	Makes the interrupt task state suspended pending a signal interrupt from an interrupt handler. Used by an interrupt task to signal its readiness to service an interrupt.
Interrupt	Enable	Enables an external interrupt level.
	Disable	Disables an external interrupt level.
	Get exception handler	Reads the location- and exception-handling mode of the current operating system exception handler for a task.
	Set exception handler	Establishes the location-and exception-handling mode of the current OSP exception handler for a task.
Segment	Create segment	Allocates dynamically an area of memory of a specified length in 16-byte paragraph units up to a maximum of 64-k bytes (for example, for use as a buffer). Returns a location token for the segment allocated.
	Delete segment	De-allocates the memory segment indicated by the parameter token.
	Enable deletion	Allows the system data type value indicated by the location token to be deleted.
	Disable deletion	Prevents the system data type value indicated by the location token from being deleted.
Mailbox	Create mailbox	Creates a mailbox with the specified task queueing discipline. Returns a location token.
	Delete mailbox	Deletes a mailbox, and returns its memory. If tasks are waiting for the mailbox, they are awakened (their state is made ready) with an appropriate exception condition. If messages are waiting for tasks, they are discarded.
	Send message	Sends a message segment to a mailbox.
	Receive message	A task is ready to receive a message at a mailbox. The task is placed on the mailbox task queue. The task may optionally wait for a response indefinitely, or a number of time intervals (generally 10 ms long), or not at all. When complete, the primitive returns to the task the location token of the message segment received.
Region	Create region	Creates a region data type value, specifying a queuing discipline. Returns a token for the region.
	Delete region	Deletes a region if and only if the region is not in use.
	Accept control	Gains control of a region if it is immediately available, but does not wait if it is not available.
	Receive control	Is the same primitive as accept control but the task that performs it may elect to wait.
	Send control	Relinquishes a region.

The 86/30 operates by creating, manipulating and deleting individual system objects. When an object is created, the 86/30 returns its name to the creating task. This name is referred to and used as an abstract data type, called a token. The token is a highly efficient way of accessing the 86/30 address space. Referring to a segment object, for example, causes a 16-bit address to be loaded into one of the processor segment registers, which can then be used to directly address a paragraph (16-byte unit) anywhere in the 1-megabyte address space. Task creation is also accomplished in this manner and requires only the specification of a priority, a task private data segment (if needed), a task stack, and a task program starting address.

To take full advantage of multiprogramming, the operating system must provide each application with a separate environment—that is, separate memory and tasks. This isolation both protects independent programs from interfering with one another and allows the application programmer to work without regard to the other application programs in the system. The 80130 supports multiprogramming with the job data type. The creation of a job requires the specification of a large number of parameters and is normally done only when the system is being initialized.

Interrupt-driven

In multitasking systems, there are two common techniques for deciding which task is to be run at any given moment. Time slicing runs tasks in rotation and is the technique used in time-sharing systems. Priority-based scheduling, on the other hand, compares assigned task priorities to decide which task is to be run next. Further, priority-based systems are usually preemptive—in other



3. Inside view. The silicon operating system is realized in the 80130 pictured here in the form of firmware containing the operating-system kernel, plus the two timers, interrupt-handling logic, and other support hardware necessary to handle complex multitasking applications.

words, a higher-priority task is executed as soon as it is ready to run, rather than only after the current task has run to completion of a time slice.

The 86/30 supports preemptive, priority-based scheduling. The built-in scheduler performs all task-scheduling functions and controls the transition of tasks among the five possible operating states. Tasks that may be executed are in what is called the ready state. The one with the highest priority will be running, while all those of lower priority remain ready.

The 86/30 hardware timer facilities support time-outs with the asleep state. Tasks can put themselves into the asleep state in one of two ways. They can wait for a predefined number of 10-ms time periods (or for some other user-defined value if 10 ms is not appropriate for the application), or they can wait for a message that a shared resource, such as an I/O device, has by now been made available.

Tasks can also make themselves or other tasks enter the nonready state called suspended. A task may be suspended more than once, and suspensions are cumulative, requiring the task to be resumed for each suspen-

sion. A task marked asleep can also be suspended. It will then be put in the special state that requires both the time-expiration condition and the suspension condition to be met before it again becomes ready.

The 80130's preemptive-scheduling algorithm ensures that the highest-priority task that is in the ready state will receive the processor and that a task that is running will continue to do so until a higher-priority interrupt occurs (including a time-out interrupt) or until it relinquishes the resources that would allow a higher-priority task to make the transition to the ready status. Each task is given a priority-and-interrupt level relative to every other task and interrupt when it is created, but task priority may also be altered dynamically.

The 80130 maps external interrupts directly into internal task priorities, using the interrupt control logic included on the chip.

Two methods of interrupt management are supported for each level. The first is the interrupt handler. It can be used for time-critical interrupts or for interrupt processing requiring a small amount of work—for example, entering an input character into a buffer. While executing in the interrupt handler, a task will be restricted to calling a very limited set of operating-system primitive functions: enter-interrupt, get-level, signal-interrupt, and exit-interrupt.

The second method, the more general interrupt task, can be used when there is more processing work to be done. This interrupt task is an ordinary 80130 task, but one that cannot be suspended. Its processing of an interrupt begins with execution of the interrupt handler for that level. It is the handler that fields the time-critical portions of the interrupt and then optionally invokes an interrupt task.

Segments, mailboxes, and regions

The 80130 also provides a free-memory manager that allocates memory to requesting tasks dynamically. This manager operates within the pool of memory resources allocated by the create-job function of the containing job. When a system object is created, the memory manager allocates the required memory; when the value is deleted, it de-allocates it. This operation is implicit, and a separate create-segment call is not required. Two related kernel primitives are provided to enable or disable the deletion of individual system objects. When a value is deleted, its memory is automatically recovered for the job memory pool. When a value is created, the deletion function is enabled, so that the disable-deletion primitive must be executed to "lock" it in memory and remove it from the available memory list.

The technique used to facilitate communication and synchronization is a mailbox, a system data type designed to pass this information reliably and efficiently between tasks. Mailboxes support both priority and first-in, first-out queues. If the receiving task is of a higher priority than the sending task, receipt of a message can cause preemptive rescheduling of the sending and receiving tasks.

One of the most difficult problems to solve in a multitasking system is mutual exclusion. Mutual exclusion is absolutely essential to a multitasking system, for it guarantees proper processing whenever two tasks require the exclusive use of a memory or I/O resource like a disk drive. Without mutual exclusion, the higher-priority task would immediately gain access to the shared resource, even if the lower-priority task is already in the process of modifying the information.

TABLE 2: EXAMPLES OF THE PERFORMANCE OF OPERATING-SYSTEM PROCESSOR PRIMITIVES

Data-type class	Primitive	Execution speed* (μs)
Job	Create job	2,950
Task	Create task (no preemption)	1,360
Segment	Create segment	700
Mailbox	Send message (with task switch)	475
	Send message (no task switch)	265
	Receive message (task waits)	540
	Receive message (message waiting)	260
Region	Send control	170
	Receive control	205

*in the 8-MHz iAPX 86/30 configuration

The 80130 solves the problem of mutual exclusion with the region object. It provides for both code regions and data regions, and both may be protected from simultaneous access by multiple tasks. If one task is in a region when another higher-priority task requests control (with receive-control), the task currently in the region will be run at the higher priority of the requester until it relinquishes the region via send-control. At that point a scheduling preemption will occur.

The performance of several 80130 primitives is given in Table 2. These times are shown for an iAPX 86/30 implementation at 8 MHz and rival those of similar functions in today's high-end minicomputers, being an entire order of magnitude faster than those of the previous generation of microprocessors. Indeed, many of these primitives operate faster than the basic multiply and divide operations of the last generation of machines.

Configuration plus

In addition to placing the 80130 in his system, the 86/30 user must supply a configuration and initialization area adjacent to the kernel store. A working area of approximately 1,500 bytes for use as the kernel stack will also be allocated in system random-access memory as part of the system initialization process.

The configuration process for the 80130 is simplified by the use of the iAPX 86/30 and 88/30 Operating-System–Processor Support Package. This software package, which runs on Intel development systems, includes optional user parameter validation code, system initialization software, and the 80130 interface library. This library and the 80130 design itself provide position independence for the 80130. All accesses to the 80130 are indirect, made through an on-chip jump table at an address supplied by the interface library. With this arrangement, the 80130 can be located on any 16-K-byte boundary in the user's address space. The on-chip jump table also ensures that if the primitives are changed, the user-to-program interface remains identical. □

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 246-7501

Printed in U.S.A./T-3020/20K/1081/BL JL